This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 592 091 A2**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: **93306582.3**

(51) Int. Cl.5: **G06F 9/46**

(22) Date of filing: **19.08.93**

(30) Priority: **06.10.92 US 957339**

(43) Date of publication of application:
13.04.94 Bulletin 94/15

(84) Designated Contracting States:
DE FR GB

(71) Applicant: **International Business Machines Corporation**
Old Orchard Road
Armonk, N.Y. 10504(US)

(72) Inventor: **Banda, Venu P.**
11411 Research Blvd., No.1012
Austin, Texas 78759(US)
Inventor: **Selby, Robert F.**
2510 Dowd Lane
Austin, Texas 78728-6739(US)
Inventor: **Snyder, Scott E.**
4503 Bridlewood
Austin, Texas 78727(US)
Inventor: **Tuatini, Jeffrey T.**
9417 Hills Trail, Apt. No.2075
Austin, Texas 78759(US)

(74) Representative: **Burt, Roger James, Dr.**
IBM United Kingdom Limited
Intellectual Property Department
Hursley Park
Winchester Hampshire SO21 2JN (GB)

(54) **Method and system for object management.**

(57) The present invention provides a method and system in a data processing system, having a multitasking operating system which includes a plurality of processes, for providing communication between objects executing within the processes in the multitasking operating system, the method and system includes registering an object within a communications manager in response to a launching of the object. The communications manager monitors all objects registered with the communications manager within the plurality of processes. A determination of whether a first object is registered is made utilising the communications manager, in response to receiving a request from a second object to send a message to the first object: Automatic initiation of the launching the first object within the processes is performed if the first object is unregistered utilising the communications manager. Next, the first process containing the first object is bound to the second process containing the second object, wherein a communications path is established between the first process and the second of the process. The message is sent to the first object from the second object via the established communications path between the two processes, wherein communication between the first object and the second object is automatically established.

EP 0 592 091 A2

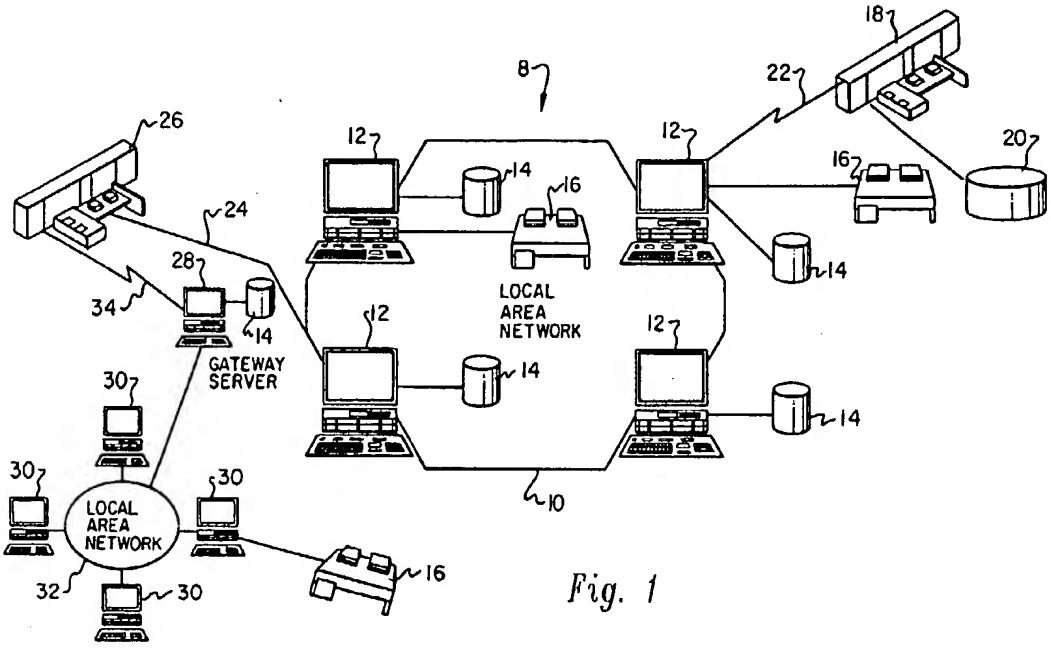
8
10
12
14
16
18
20
22
24
26
28
30
32
34
LOCAL AREA NETWORK
GATEWAY SERVER
LOCAL AREA NETWORK

Fig. 1

WEST

Technical Field

The present invention relates in general to a method and system for managing communications in a data processing system and in particular to a method and system for managing communications between objects executing different processes in a data processing system.

Background Art

In multi-tasking operating systems, hardware may be managed as a shared resource to be distributed among concurrently executing applications or programs. In a multi-tasking operating system, resources such as the processor, memory, files, devices and inter-process communications structures are also shared in the operating system among concurrently executing applications, programs, or objects. "Timeslicing" is utilised to share the processor with various concurrently executing applications or objects. Generally, an operating system will switch between applications allowing each application to run for a short period of time, a "timeslice". Thus, in this manner, the processor may be shared between various applications or objects, enabling multitasking.

For example, OS/2 is an operating system that offers a multitasking architecture, providing the capability to execute multiple programs in a protected environment. OS/2 is a trademark and a product of International Business Machines Corporation. OS/2 includes a hierarchy of multi-tasking options called sessions, processes, and threads. A "session" is a unit of user input/output ("I/O") device sharing. A "process" is a unit for sharing for various resources, such as memory, files, semaphores, queues, and threads. A process may contain multiple programs or objects. More information on OS/2 can be found in a text entitled The Design of OS/2 by Deitel and Kogan, published by Addison-Wesley Publishing Co. 1992.

Objects are DCE objects as defined in OSF DCE User's Guide And Reference from The Open Software Foundation Inc. located at 11 Cambridge Center, Cambridge, Massachusetts 02143. In an environment containing different and potentially independent application objects (all of which are built upon a base class hierarchy), which are being developed, a need exists to decouple the run-time environments in which the objects execute. The need for decoupling arises out of a concern for integrity between the objects. Generally, if two objects are executing within the same process, the failure of one object will jeopardise the other object. In other words, if one object fails within a process, normally all of the other objects within the same process will also fail or become unstable. An erroneous implementation of one object also may corrupt the other objects.

Objects executing within the same process are able to easily exchange data. Frequently, it is desirable to interchange data or send messages to an object within a different process. Data exchange becomes more difficult between objects executing in different processes.

Presently, data sharing and message sending between two objects, located in different processes, are handled by writing a protocol specific to the two objects. As a result, data sharing and message sending may only occur between those two objects. In order to add a third object, additional code must be produced to allow for data sharing and message sending between all three objects.

Therefore, it would be desirable to have a method and system to manage the distribution and communication of data and messages across processes without having to produce new protocols for each additional process.

Disclosure of the Invention

It is therefore one object of the present invention to provide a method and system for managing communications in a data processing system.

It is another object of the present invention to provide a method and system for managing communications between objects executing within a data processing system.

It is yet another object of the present invention to provide a method and system for managing communications between objects executing in different processes within a data processing system.

The foregoing objects are achieved as is now described. The present invention provides a method and system in a data processing system, having a multitasking operating system that includes a plurality of processes, for providing communication between objects executing within the processes in the multitasking operating system. The method and system includes registering an object with a communications manager in response to a launching of the object. The communications manager monitors all objects registered to it within the plurality of processes. A determination of whether a first object is registered is made utilising the communications manager, in response to receiving a request from a second object to send a message to the first object. Automatic initiation of the launching the first object within the processes is performed utilising the communications manager if the first object is unregistered. Next, the process containing the first object is bound to the process containing the second object, wherein a communications path is established between the two processes. The mes-

WEST

sage is sent to the first object from the second object via the communications path between the two processes, wherein communication between the first object and the second object is automatically established.

Brief Description of the Drawings

The invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

Figure 1 depicts a pictorial representation of a data processing system which may be utilized to implement a method of the present invention;

Figure 2 is a diagram illustrating the establishment of a communications link between an object dispatch process (ODP) and a process in accordance with a preferred embodiment of the present invention;

Figure 3 depicts a data flow diagram illustrating the establishment of a communications link between two processes in a situation where one object has not been instantiated in accordance with a preferred embodiment of the present invention;

Figures 4A and 4B are logical flowchart for creating a communications path in accordance with a preferred embodiment of the present invention; and

Figure 5 depicts a logical flowchart of the process followed by a message dispatcher (MD) in accordance with a preferred embodiment of the present invention.

Detailed Description of the Invention

With reference now to the figures, and in particular with reference to Figure 1, there is depicted a pictorial representation of a data processing system 8 that may be utilised to implement a method and system of the present invention. As may be seen, data processing system 8 may include a plurality of networks, such as local area networks (LAN) 10 and 32, each of which preferably includes a plurality of individual computers 12 and 30, respectively. Of course, those skilled in the art will appreciate that a plurality of intelligent work stations (IWS) coupled to a host processor may be utilised for each such network.

As is common in such data processing systems, each individual computer may be coupled to a storage device 14 and/or a printer/output device 16. One or more such storage devices 14 may be utilised, to store documents or resource objects which may be periodically accessed by any user within data processing system 8. In a manner well known in the prior art, each such document or resource object stored within a storage device 14 may be freely interchanged throughout data processing system 8 by transferring a document to a user at an individual computer 12 or 32, for example.

Still referring to Figure 1, it may be seen that data processing system 8 may also include multiple mainframe computers, such as mainframe computer 18, which may be preferably coupled to LAN 10 by means of communications link 22. Mainframe computer 18 may also be coupled to a storage device 20 which may serve as remote storage for LAN 10. Similarly, LAN 10 may be coupled via communications link 24 through a subsystem control unit/communications controller 26 and communications link 34 to a gateway server 28. Gateway server 28 is preferably an individual computer or IWS which serves to link LAN 32 to LAN 10.

As discussed above with respect to LAN 32 and LAN 10, a plurality of documents or resource objects may be stored within storage device 20 and controlled by mainframe computer 18, as Resource Manager or Library Service for the resource objects thus stored. Of course, those skilled in the art will appreciate that mainframe computer 18 may be located a great geographic distance from LAN 10 and similarly LAN 10 may be located a substantial distance from LAN 32. For example, LAN 32 may be located in California while LAN 10 may be located within Texas and mainframe computer 18 may be located in New York.

A multitasking environment including multiple processes may be found on individual computers 12 and 30, on gateway server 28, on some computer in LAN 10 or 32, or on mainframe computer 18.

An object management system is utilized to manage communications across processes and to track and locate objects executing within the processes in the multitasking environment. The object management system includes an object dispatcher process (ODP) and a message dispatcher (MD). This management system may include: (1) a launching mechanism to launch or initiate the launching of an object within a process, (2) a mechanism for grouping objects within a process, (3) a location mechanism to locate the process within which an object is executing, and (4) a system for forwarding messages.

An object is identified by a static identity and a dynamic identity. The "static identity" is the DCE name of the object, i.e., its spatial identification. For example, /.../austin.ibm.com/host/obj1 identifies "obj1" as an object in the "host" subdirectory of "austin.ibm.com" cell. Next, the "dynamic identity" of an object is defined by a process-ID and a pointer.

WEST

The object management system monitors the objects within the various processes and maintains a local registry of all active objects. An object requesting a communications path to a target object utilizes a static identity to identify the target object. An object locator component of the object management system in the ODP is utilized to identify the dynamic identity of an object that corresponds to the static identity of the object. A mapping table is employed in the ODP to track and monitor objects registered to the ODP. If an entry for the name of an object is absent from the mapping table, the ODP launches or initialises the launching of the object. The ODP registers an object when the ODP launches the object. Registration also occurs when an object communicates with the ODP.

Three pieces of information are required to launch an object: (1) object identity, (2) class of the object, and (3) object handler. With an object identity, the static identity is sufficient if the object is not yet active. The object handler includes the code corresponding to the methods of the object.

The ODP is a pivotal process in the method and system utilised to initiate and manage inter-object communication across processes and also may provide an "object locator" service. The ODP is a communications manager and constantly monitors for messages on a well-known port. Each communications path or channel to the ODP from a process is serviced by a "receiver thread", which is responsible for receiving a message on the appropriate path, decoding the message, and taking or initiating appropriate actions. A "receiver thread" is a thread within the ODP. Appropriate actions such as launching or initiating the launching of objects and creating communication paths or channels between processes are examples of actions taken by the receiver thread.

A message received by the ODP may include the following components: (1) object name, (2) class, (3) operation, and (4) arguments. An operation may be, for example, to find and load an object, to create a new instance of an object, or may be any other operation that is defined in an object's object handler. The object name is the name of the object that is to receive a message. The class is the class of that object. ODP registers objects that it launches and objects that communicate with the ODP.

The MD may manage inter-process and intra-process communication between objects. It is responsible for delivering messages or methods to objects within the process in which the MD is located. Incoming messages from another process are handled by forwarding them to the appropriate object by utilizing the object handle field within the message.

When a local object sends a message to a foreign object, located in a different process and no communications path has been established between the two processes, the MD sends the message to the ODP to establish a communications path to the process containing the foreign object. The ODP searches the mapping table to determine if an entry for the foreign object exists.

In the event that an entry exists, the ODP extracts the dynamic identity (process-ID and handle within the process) of the foreign object from the mapping table. The handle within a process is utilised to identify an object within the process. The ODP sends the message to the foreign object and returns a message to the requesting process that contains the method name, argument list, and a handle within the process for the foreign object. The MD of the local object utilises this information to send a message to the foreign object via a communications path or channel.

If the ODP does not find an object entry in the mapping table, the ODP searches a class table to determine whether or not the foreign object's class has been loaded into any process. If it has, a message is sent to that process to launch the foreign object. The message sent to the process may be in the format: (operation, argument list, class name, a DLL name). DLL stands for Dynamic Link Library. If the class of the foreign object has not been loaded into a process, a new process is started and the foreign object is launched within the new process.

In launching a object, it is assumed that the DCE name of the object is known as well as the class of the object and the name of the object handler for the object. The object handler of the object is the name of the dynamic link library (DLL) file in OS/2. A DLL is a set of subroutines that are dynamically bound to the calling program or object when the program or object is loaded into memory or when they are loaded explicitly by an already executing program or object. The DLL file contains the methods that the foreign object will respond to. More information on DLL calls may be found in LaGarde et al., IBM Operating System/2 Version 1.2 Programming Guide, International Business Machines Corporation, Document No. 00F8833, 1989 and in a text entitled The Design of OS/2 by Deitel and Kogan, published by Addison-Wesley Publishing Co. 1992.

In the situation where a communications path has already been established between the process containing a local object and the process containing the foreign object, the invocation of an operation on a foreign object by a local object results in the MD constructing the necessary message and forwarding it directly to the process that is executing the foreign object over the communications

WEST

path that was previously set up. MD maintains two tables for managing inter-process and intra-process communications: (1) class table and (2) object table. The class table is a data structure that maintains a list of all classes that are registered in a particular process. The object table is a table for storing a list of all objects existing within a particular process and foreign objects that are in communication with these objects. As a result, MD also provides a grouping mechanism to group objects within a process.

Foreign objects are associated with binding information and channel identification that is necessary for reaching a foreign object. Channel identification is data that uniquely identifies a communications channel between a sender and a receiver. For example if a "sockets" implementation is being utilised for communication, a port number would uniquely identify the communications channel and if "shared memory" is employed for communication, the name of the shared memory segment would uniquely identify the communications channel. The MD also continuously "pings" or monitors the ODP to see if the ODP is currently executing. If MD determines through a time out mechanism that ODP is no longer running, it initiates a "clean up" and terminates itself. Referring now to Figure 2, there is depicted a diagram illustrating the establishment of a communications link between an ODP and a process. ODP 200 scans port 202 for requests from other processes or subsystems, such as process 204, to connect to ODP 200. A subsystem is a logical subcomponent of a larger system, e.g., a file system in a operating system. Port 202 is a port known to those skilled in the art and may be associated with a well known server process (i.e., name server, file-transfer protocol server, etc.). "Known" ports are those that are active on all machines. In a network of computers, a process on a machine may be identified by a pair: (machine-ID, port number). The port number identifies a process on a given machine.

In response to such a request, ODP 200 returns a "binding" to the requesting process, process 204. The new binding represents a new communications path or channel, channel 206, between the process 204 and ODP 200. ODP 200 utilises this channel in all subsequent interaction with process 204.

As mentioned before, each of the communications paths or channels associated with the ODP are serviced by a "receiver" thread. This thread is responsible for receiving the message on the appropriate path or channel, decoding the message, and taking or initiating the appropriate action. Messages received by ODP include object name, class, operation, and arguments.

With reference now to Figure 3 , there is depicted a data flow diagram illustrating the establishment of a communications link between two processes in a situation where one object has not been instantiated. Object 250 of class A is executing in process 252. Object 250 attempts to instantiate a new object, Obj2, of class B. MD 254 searches it's local class table and does not find class B.

Consequently, MD 254 sends a message to ODP 256 via communications path 258, asking ODP 256 to instantiate Obj2 of class B. Communications path 258 may be established as previously illustrated in Figure 2. ODP 256 decodes the message and looks in its mapping table 257 for Obj2. Assuming ODP 256 does not find Obj2 in mapping table 257, ODP 256 then obtains the name of Obj2's class from the message.

ODP 256 initiates process 260 with class B loaded in it. MD 261 in process 260 instantiates or launches Obj2 and returns Obj2_Handle and a new binding to ODP 256 to form communications path 262. ODP 256 updates mapping table 257 to include the association between Obj2 and Obj2_Handle.

ODP 256 returns binding to MD 254 to establish communications path 264. The MD 254 updates its local object table to associate Obj2 with the communications path to process 260 and returns a local handle for Obj2 to Object 250, forming communications path 264. All subsequent operations on Obj2 by objects in process 252 are sent directly to process 260 over communications path 264.

Referring now to Figure 4A and Figure 4B, there is depicted a logical flowchart for creating a communications path. The procedure begins as illustrated in block 400 and thereafter proceeds to block 402, which depicts the sending of a request, M0, to the ODP by a requesting object, which is the client of the ODP, to create a communications path to a target object. M0 is a message, containing the following information on the target object: (1) object name, (2) object class, (3) method name, (4) an argument list, and (5) a DLL file name. M0 identifies the object utilising a static identity. The procedure then proceeds to block 404, which illustrates the reception of the message, M0, by the ODP, the decoding of the message, M0, and the looking up of the object name in a mapping table by ODP.

Next, the procedure then proceeds to block 406, which depicts a determination of whether or not an object name has been found in the ODP's mapping table corresponding to the object name specified in M0. The mapping table contains an entry for each object that is registered with the ODP. Each entry contains: (1) object name, (2)

WEST

process-ID, and (3) handle within the process in which the object is found. As mentioned before, objects are registered when they send a request to the ODP or when the ODP launches an object. If an object name has been found, the procedure then proceeds to block 408, which illustrates the extraction of a tuple (process-ID, handle within process) from the mapping table and corresponding the tuple to the object name in M0.

Afterward, the procedure proceeds to block 410, which depicts the sending of a message, M1, to the MD in the process containing the target object. The message, M1, contains the method name, an arguments list, and the handle within the process. M1 contains the dynamic identity of the object. The procedure then proceeds to block 414, which illustrates the MD in the process of the target object receiving the message. The message includes the method name, an argument list, and the handle within the process for the object requested by the requesting object. Thereafter, the procedure proceeds to block 430 via connector C, which illustrates the return of a binding by the ODP to the client for the process containing the target object. Binding information is found within a connection table in the ODP. The connection table includes a process-ID associated with binding information for a particular process. This connection table is utilised to provide binding information to various objects requiring communication's paths to objects in other processes. The process thereafter terminates as illustrated in block 432.

Referring back to block 406, if an object name is not found in the mapping table, the procedure proceeds to block 416, which illustrates the searching of a class table for the object class of the target object requested. The procedure then proceeds to block 418, which depicts a determination of whether or not the class is found in the class table in the ODP. The class table contains information on the object class and the process-ID associated with each object class. A class is registered when an object contacts the ODP if the class is not already in the class table. A class is also registered if the ODP launches or initiates the launching of a new process having a class not found in the ODP's class table. If the class is found, the procedure proceeds to block 419, which illustrates the extraction of a class-ID for the target object's class from the class table. Thereafter, the procedure continues to block 420 in Figure 4B via connector A. Block 420 illustrates the extraction of the process-ID in which the class of the target object is loaded. Thereafter, the procedure proceeds to block 422, which depicts the sending of a message, M2, to the process in which the class of the target object is loaded. The message, M2, contains (1) the object class, (2) opn = new, and (3) the DLL of the target object. M2 is a request to launch the target object in the process. Next, the procedure proceeds to block 424, which illustrates waiting for a reply from the process in which the class of the target object is loaded.

Thereafter, the procedure proceeds to block 426, which depicts the reception the process-ID and the handle of the newly created target object from the process in which the class is loaded. The procedure then proceeds to block 428, which illustrates the registration of the target object and its handle into the mapping table by the ODP. The procedure proceeds to block 430, which depicts the ODP returning a "binding" to the process of the newly created target object. As a result, future requests for the target object by the requesting object may be directly sent to the process containing the target object, and thus, bypassing the ODP and thereby increasing performance and speed of communications between objects. The procedure then terminates as illustrated in block 432.

Referring back to block 418, if a class is not found in the class table, the procedure then proceeds, via connector B, to block 433, which illustrates the initiation or launching of a new process and the loading of the class of the target object into the new process utilising the class name extracted from the request from the requesting object. The procedure advances to block 434, which depicts the registration of the class name and the process-ID of the new process into the class table in the ODP.

Thereafter, the procedure proceeds to block 422, which depicts the sending of a message, M2, to the new process. As described above, M2 contains the target object class, opn = "new", and the DLL of the target object. The procedure next advances to block 424, which illustrates waiting for a reply from the process. Afterward, the procedure continues to block 426, which depicts the returning of the process-ID and the handle of the newly created target object by the process to the ODP. The procedure then proceeds to block 428, which illustrates the registration of the target object and its handle into the mapping table by the ODP. The procedure next proceeds to block 430, which depicts the return of a "binding" for the process to the requesting object by the ODP. Again, the procedure then terminates as depicted in block 432.

Referring now to Figure 5, there is depicted a logical flowchart of the process followed by a message dispatcher (MD). As illustrated the procedure begins in block 500 and thereafter proceeds to block 502, which depicts the reception of a message and the decoding of the message by the MD. Thereafter, the procedure proceeds to block 504, which illustrates a determination of whether or not opn equals "new". If opn does not equal "new" the

WEST

procedure advances to block 506, which depicts the extraction of the handle, method name, and argument list from the message and utilising a system object model (SOM) dispatch to deliver the message to the object. SOM is a mechanism utilised to deliver messages to objects within the same process. SOM is found in the OS/2 2.0 Software Development Toolkit (SDK) available from International Business Machines Corporation.

Thereafter, the procedure proceeds to block 508, which illustrates the returning of the results to the object sending the message. The procedure then terminates as depicted in block 510.

Referring again to block 504, if "opn" equals "new", the procedure then proceeds to block 512, which illustrates using the object DLL to load the class into the process. Thereafter, the procedure proceeds to block 514, which depicts the invocation of SOM dispatch on the above "class object" to create a new object. The procedure proceeds to block 516, which illustrates the sending of a reply to ODP to register the newly created object into ODP's mapping table. The procedure again terminates as illustrated in block 510.

Claims

1. A method for use in a data processing system (8), having a multitasking operating system which includes a plurality of processes (252, 260), for providing communication between a plurality of objects (250, Obj2) executing within said plurality of processes in said multitasking operating system, said method comprising the steps of:

   registering (402) an object (250) with a communications manager (256) in response to a launching of said object by an object manager (256);

   determining (406) whether a first object (Obj2) is registered with said communications manager, in response to receiving a request from a second object (250) within said plurality of processes to send a message to said first object;

   automatically initiating launching (433) of said first object within said plurality of processes utilising said communications manager if said first object is unregistered;

   binding (430) a first (260) of said plurality of processes containing said first object (Obj2) to a second (252) of said plurality of processes containing said second object (250), wherein a communications path (264) is established between said first of said plurality of processes and said second of said plurality of processes; and

   sending said message to said first object (Obj2) within said first (260) of said plurality of processes from said second object (250) in said second (252) of said plurality of processes via said established communications path (264) between said first of plurality of processes and said second of plurality of processes, wherein communication between said first object and said second object is automatically established.

2. A method as claimed in claim 1 further comprising registering an object (250) in response to a request for communication with said communications manager (256) by said object.

3. A method as claimed in claim 2, wherein said communications manager (256) registers an object (250) by storing data for an object within a mapping table (257), wherein said mapping table contains an entry for each object registered by said communications manager.

4. A method as claimed in claim 3, wherein said entry includes an identifier identifying the process (252) in which said registered object (250) has been launched.

5. A method as claimed in claim 4, wherein said entry further includes a pointer to said object (250).

6. A method as claimed in claim 5, wherein said binding (430) of said first (260) of said plurality of processes to said second (252) of said plurality of processes includes sending said second of said plurality of processes binding data for said first of said plurality of processes.

7. A method as claimed in claim 6, wherein each of said plurality of processes includes a message dispatcher (254, 261) for managing communications within a process and between processes, wherein said message dispatcher has a communications path (262) to said communications manager (256).

8. A method as claimed in claim 7, wherein said message dispatcher (254, 261) for a process (252, 260) maintains a data structure having a list of all objects (250, Obj2) within said process and all objects in other processes having a communications path (264) to said process.

9. A method as claimed in any previous claim, wherein said communications manager (256) monitors all objects (250) registered to said communications manager within said plurality of processes (252, 260); and further compris-

WEST

ing:

prior to automatically initiating launching (433) of said first object, automatically searching (416) said plurality of processes for a process containing a class for said first object within said plurality of processes if said first object is unregistered and if a process containing said class is present.

10. A data processing system (8), having a multitasking operating system which includes a plurality of processes (252, 260), for providing communication between a plurality of objects (250, Obj2) executing within said plurality of processes in said multitasking operating system, said data processing system comprises:

registration means (402) for registering an object (250) with a communications manager (256) in response to a launching of said object by said object manager (256);

determining means (406) for determining whether a first object (Obj2) is registered with said communications manager in response to receiving a request from a second object (250) within said plurality of processes to send a message to said first object;

initiation means (433) for automatically initiating launching of said first object within said plurality of processes utilising said communications manager if said first object is unregistered;

binding means (430) for binding a first (260) of said plurality of processes containing said first object (Obj2) to a second (252) of said plurality of processes containing said second object (250), wherein a communications path (264) is established between said first of said plurality of processes and said second of said plurality of processes; and

sending means for sending said message to said first object (Obj2) within said first (260) of said plurality of processes from said second object (250) in said second (252) of said plurality of processes via said established communications path (264) between said first of plurality of processes and said second of plurality of processes, wherein communication between said first object and said second object is automatically established.

WEST

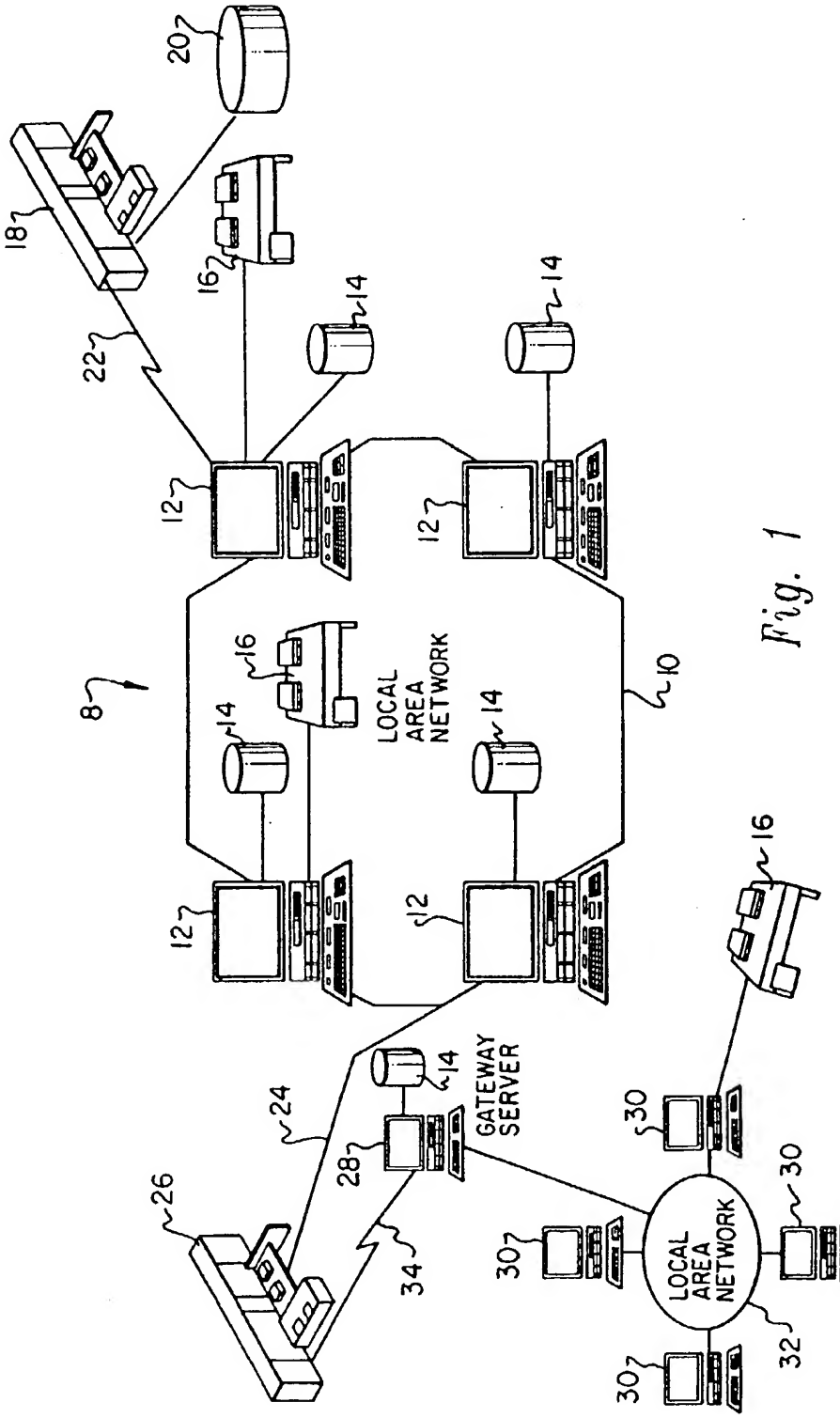
20
18
16
22
14
14
12
12
8
16
LOCAL AREA NETWORK
14
14
10
12
12
16
14
GATEWAY SERVER
24
28
30
26
34
30
30
LOCAL AREA NETWORK
32
30

Fig. 1

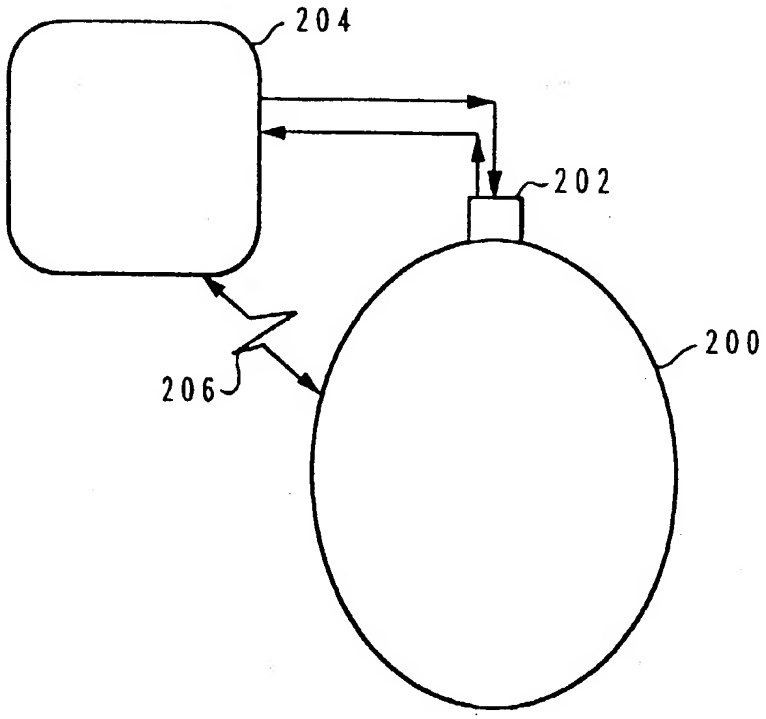

Fig. 2

WEST

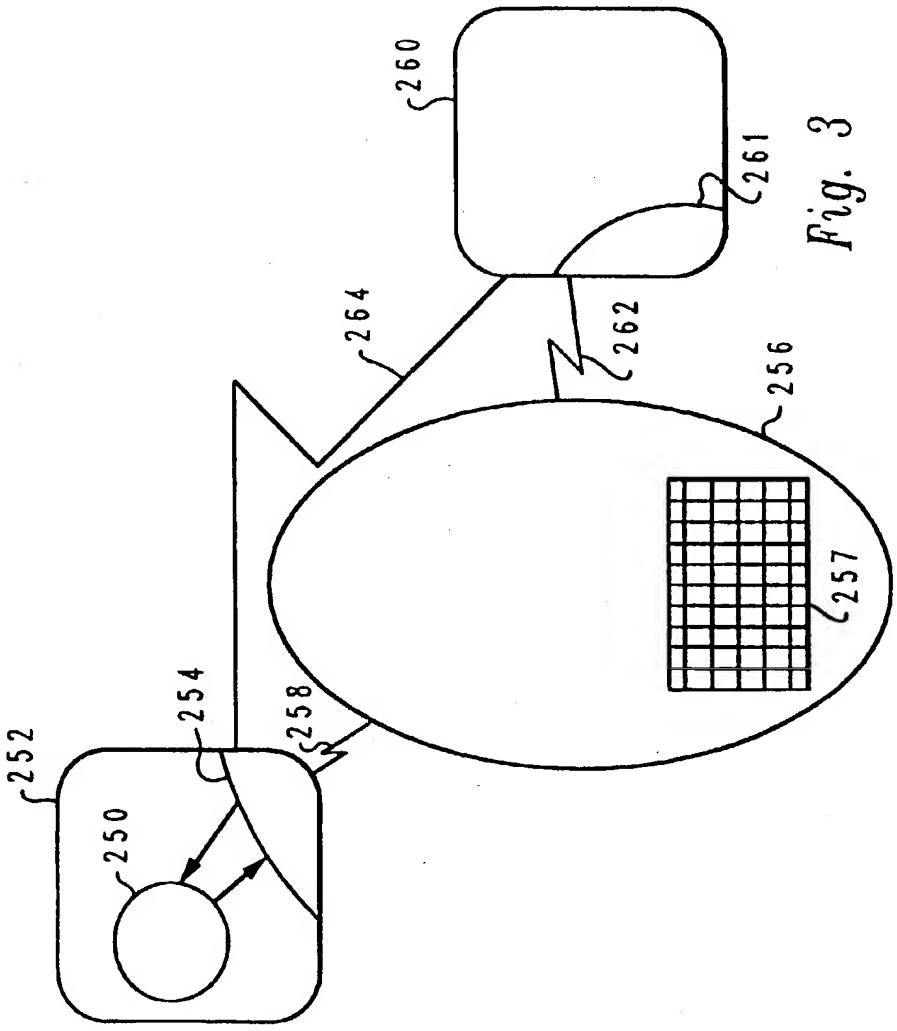

Fig. 3

WEST

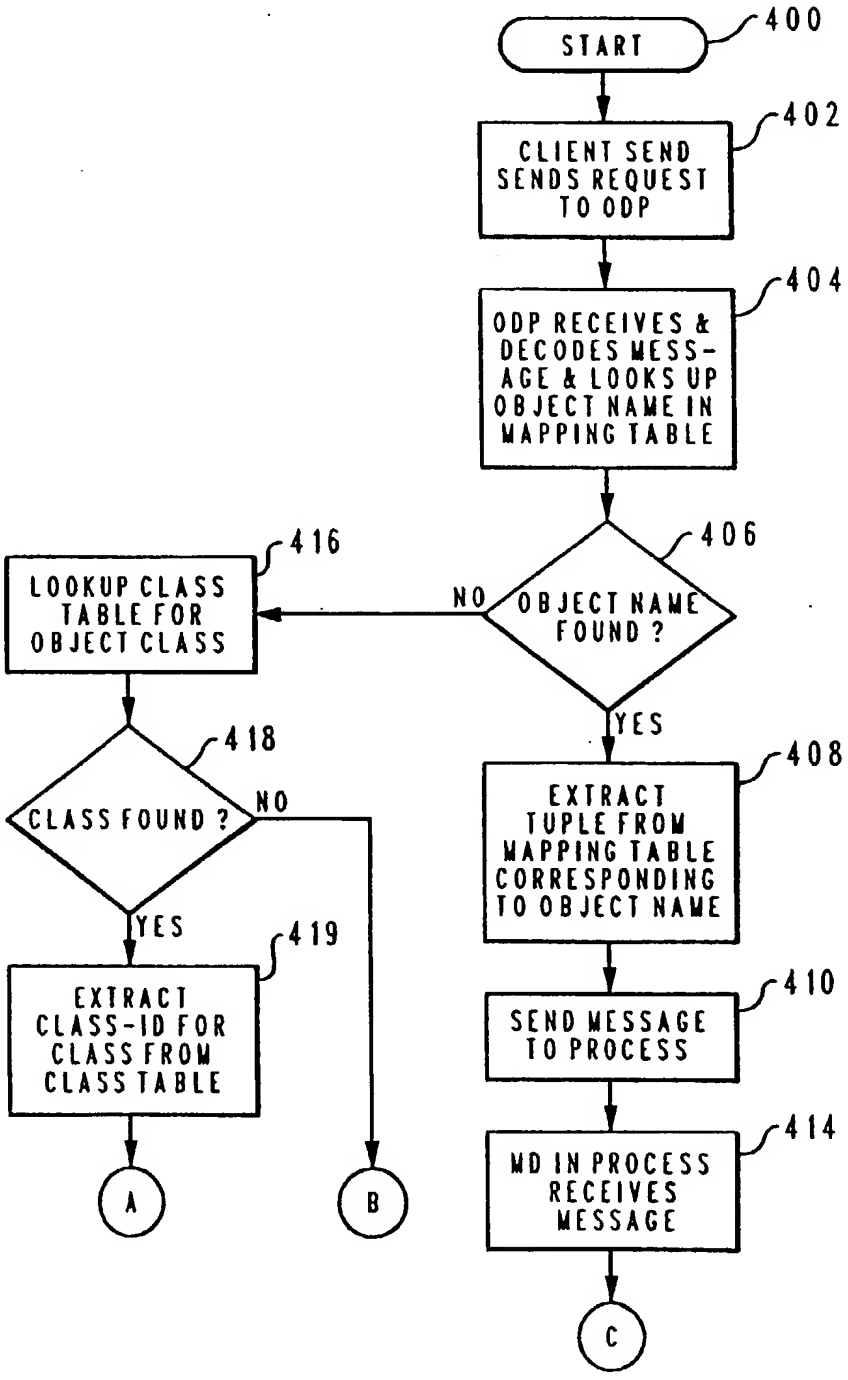
START
400
CLIENT SEND SENDS REQUEST TO ODP
402
ODP RECEIVES & DECODES MESS-AGE & LOOKS UP OBJECT NAME IN MAPPING TABLE
404
OBJECT NAME FOUND ?
406
NO
YES
LOOKUP CLASS TABLE FOR OBJECT CLASS
416
CLASS FOUND ?
418
NO
YES
EXTRACT CLASS-ID FOR CLASS FROM CLASS TABLE
419
EXTRACT TUPLE FROM MAPPING TABLE CORRESPONDING TO OBJECT NAME
408
SEND MESSAGE TO PROCESS
410
MD IN PROCESS RECEIVES MESSAGE
414
A
B
C

Fig. 4A

WEST

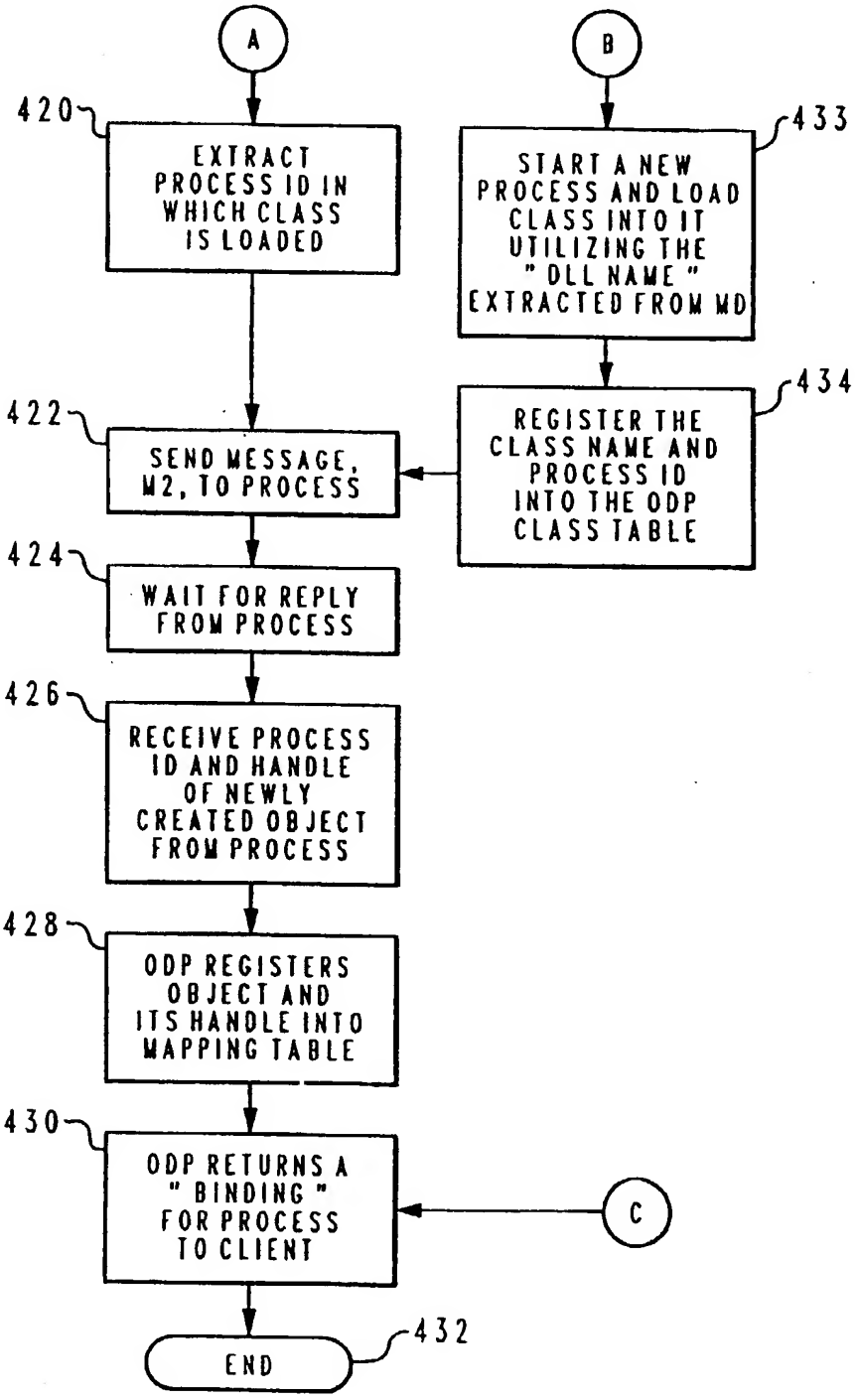
A
B
420
EXTRACT PROCESS ID IN WHICH CLASS IS LOADED
433
START A NEW PROCESS AND LOAD CLASS INTO IT UTILIZING THE " DLL NAME " EXTRACTED FROM MD
434
REGISTER THE CLASS NAME AND PROCESS ID INTO THE ODP CLASS TABLE
422
SEND MESSAGE, M2, TO PROCESS
424
WAIT FOR REPLY FROM PROCESS
426
RECEIVE PROCESS ID AND HANDLE OF NEWLY CREATED OBJECT FROM PROCESS
428
ODP REGISTERS OBJECT AND ITS HANDLE INTO MAPPING TABLE
430
ODP RETURNS A " BINDING " FOR PROCESS TO CLIENT
C
END
432

Fig. 4B

WEST

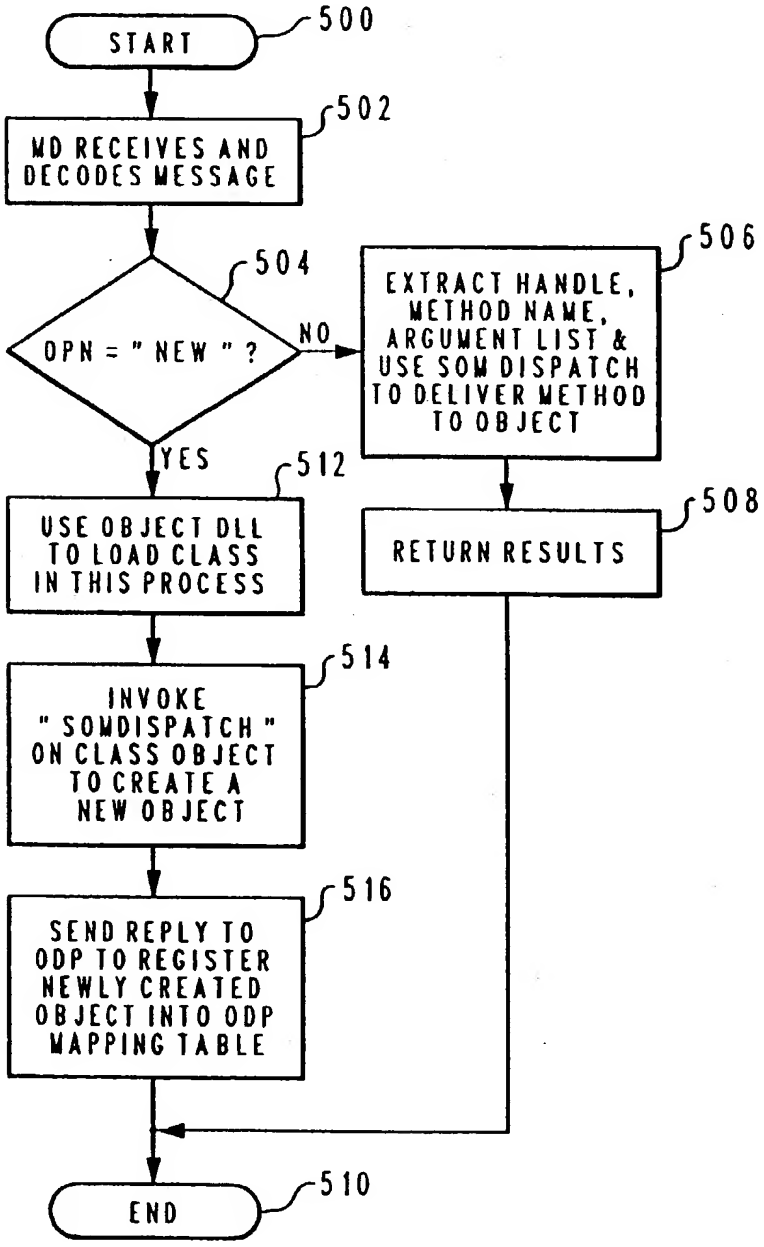
START
500
MD RECEIVES AND DECODES MESSAGE
502
OPN = " NEW " ?
504
NO
YES
EXTRACT HANDLE, METHOD NAME, ARGUMENT LIST & USE SOM DISPATCH TO DELIVER METHOD TO OBJECT
506
RETURN RESULTS
508
USE OBJECT DLL TO LOAD CLASS IN THIS PROCESS
512
INVOKE " SOMDISPATCH " ON CLASS OBJECT TO CREATE A NEW OBJECT
514
SEND REPLY TO ODP TO REGISTER NEWLY CREATED OBJECT INTO ODP MAPPING TABLE
516
END
510

Fig. 5

WEST